Midterm next week, due Friday (more on this in 1 min)

Assignment 6 due Friday before fall break

Midterm

Download from course web page when you're ready to take it (available by end of day Monday)

2 hours to complete

Must hand-in (or e-mail in) by 11:59pm Friday Oct. 7

Can use: class notes, your notes, the book, your assignments and Wikipedia.

You may not use: your neighbor, anything else on the web, etc.
What can be covered

- Anything we’ve talked about in class
- Anything in the reading (these are not necessarily the same things)
- Anything we’ve covered in the assignments

Midterm topics

- Machine learning basics
  - different types of learning problems
  - feature-based machine learning
  - data assumptions/data generating distribution
- Classification problem setup
- Proper experimentation
  - train/dev/test
  - evaluation/accuracy/training error
  - optimizing hyperparameters

Midterm topics

- Learning algorithms
  - Decision trees
  - K-NN
  - Perceptron
  - Gradient descent
- Algorithm properties
  - training/learning
  - rational/why it works
  - classifying
  - hyperparameters
  - avoiding overfitting
  - algorithm variants/improvements
- Geometric view of data
  - distances between examples
  - decision boundaries
- Features
  - example features
  - removing erroneous features/picking good features
  - challenges with high-dimensional data
  - feature normalization
- Other pre-processing
  - outlier detection
Midterm topics

Comparing algorithms
- \( n \)-fold cross validation
- leave one out validation
- bootstrap resampling
- \( t \)-test

Imbalanced data
- evaluation
  - precision/recall, \( F_1 \), AUC
  - subsampling
  - oversampling
  - weighted binary classifiers

Multiclass classification
- Modifying existing approaches
  - Using binary classifier
    - OVA
    - AVA
    - Tree-based
  - Micro- vs. macro-averaging

Ranking
- using binary classifier
- using weighted binary classifier
- evaluation

Gradient descent
- 0/1 loss
- Surrogate loss functions
- Convexity
- Minimization algorithm
- Regularization
  - Different regularizers
  - \( p \)-norms

Misc
- Good coding habits
- JavaDoc

Midterm general advice

2 hours goes by fast!
- Don’t plan on looking everything up
- Lookup equations, algorithms, random details
- Make sure you understand the key concepts
- Don’t spend too much time on any one question
- Skip questions you’re stuck on and come back to them
- Watch the time as you go

Be careful on the T/F questions

For written questions
- Think before you write
- Make your argument/analysis clear and concise
How many have you heard of?

(Ordinary) Least squares
Ridge regression
Lasso regression
Elastic regression
Logistic regression

Model-based machine learning

1. pick a model
   \[ 0 = b + \sum_{j=1}^{m} w_j f_j \]

2. pick a criteria to optimize (aka objective function)
   \[ \sum_i I[y_i(w \cdot x_i + b) \leq 0] \]

3. develop a learning algorithm
   \[ \arg\min_w \sum_i I[y_i(w \cdot x_i + b) \leq 0] \quad \text{Find } w \text{ and } b \text{ that minimize the 0/1 loss} \]

Model-based machine learning

1. pick a model
2. pick a criteria to optimize (aka objective function)
3. develop a learning algorithm

Exponential:
\[ l(y, y') = \exp(-yy') \]

Squared loss:
\[ l(y, y') = (y - y')^2 \]

Surrogate loss functions

0/1 loss:
\[ l(y, y') = I[yy' \leq 0] \]

Hinge:
\[ l(y, y') = \max(0, 1 - yy') \]

Exponential:
\[ l(y, y') = \exp(-yy') \]
Finding the minimum

You're blindfolded, but you can see out of the bottom of the blindfold to the ground right by your feet. I drop you off somewhere and tell you that you're in a convex shaped valley and escape is at the bottom/minimum. How do you get out?

Gradient descent

- pick a starting point \( \mathbf{w} \)
- repeat until loss doesn't decrease in any dimension:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)
    \[
    w_j = w_j - \eta \frac{d}{d w_j} \text{loss}(\mathbf{w})
    \]

Perceptron learning algorithm!

repeat until convergence (or for some \( \# \) of iterations):
for each training example \((f_1, f_2, \ldots, f_m, \text{label})\):

\[
\text{prediction} = b + \sum_j w_j f_j
\]

- if \( \text{prediction} \cdot \text{label} \leq 0 \); // they don't agree
  for each \( w_j \)
  \[
  w_j = w_j + f_j \cdot \text{label}
  \]
  \[
  b = b + \text{label}
  \]

\[
\begin{align*}
  w_j &= w_j + \eta y_i x_i \exp(-y_i (\mathbf{w} \cdot \mathbf{x}_i + b)) \\
  \text{or} \\
  w_j &= w_j + \eta y_i c \quad \text{where} \quad c = \eta \exp(-y_i (\mathbf{w} \cdot \mathbf{x}_i + b))
\end{align*}
\]

The constant

\[
\begin{align*}
  c &= \eta \exp(-y_i (\mathbf{w} \cdot \mathbf{x}_i + b)) \\
  \text{learning rate} & \quad \text{label} & \quad \text{prediction}
\end{align*}
\]

When is this large/small?
The constant

\[ c = \eta \exp(-y_i(w \cdot x_i + b)) \]

If they’re the same sign, as the predicted gets larger there update gets smaller

If they’re different, the more different they are, the bigger the update

One concern

\[ \arg\min_{w,b} \sum \exp(-y_i(w \cdot x_i + b)) \]

We’re calculating this on the training set

We still need to be careful about overfitting!

The min \(w,b\) on the training set is generally NOT the min for the test set

How did we deal with this for the perceptron algorithm?

Overfitting revisited: regularization

A regularizer is an additional criterion to the loss function to make sure that we don’t overfit

It’s called a regularizer since it tries to keep the parameters more normal/regular

It is a bias on the model that forces the learning to prefer certain types of weights over others

\[ \arg\min_{w,b} \sum \text{loss}(yy') + \lambda \text{regularizer}(w,b) \]

Regularizers

\[ 0 = b + \sum w_j f_j \]

Should we allow all possible weights?

Any preferences?

What makes for a “simpler” model for a linear model?
Regularizers

Generally, we don’t want huge weights
If weights are large, a small change in a feature can result in a large change in the prediction
Also gives too much weight to any one feature
Might also prefer weights of 0 for features that aren’t useful

\[ 0 = b + \sum_{j=1}^{n} w_j f_j \]

Common regularizers

- Sum of the weights: \( r(w, b) = \sum_{w_j} |w_j| \)
- Sum of the squared weights: \( r(w, b) = \sqrt{\sum_{w_j} |w_j|^2} \)

What’s the difference between these?

Common regularizers

- Sum of the weights: \( r(w, b) = \sum_{w_j} |w_j| \)
- Sum of the squared weights: \( r(w, b) = \sqrt{\sum_{w_j} |w_j|^2} \)

Squared weights penalizes large values more
Sum of weights will penalize small values more

\[ \text{argmin}_{w,b} \sum_{i=1}^{\lambda} \text{loss}(y_i^\prime) + \lambda \text{regularizer}(w,b) \]
**p-norm**

- Sum of the weights (1-norm): \( r(w, b) = \sum w_j \)
- Sum of the squared weights (2-norm): \( r(w, b) = \sqrt{\sum w_j^2} \)

\[ p\text{-norm } r(w, b) = \left\{ \begin{array}{ll} \sum w_j & \text{for } p = 1 \\ \sqrt{\sum w_j^p} & \text{for } p = 2 \\ \end{array} \right. \]

Smaller values of \( p \) (\( p < 2 \)) encourage sparser vectors
Larger values of \( p \) discourage large weights more

---

**Model-based machine learning**

1. **pick a model**
   \[ 0 = b + \sum_{j=1}^{n} w_j f_j \]
2. **pick a criteria to optimize (aka objective function)**
   \[ \sum_{i=1}^{n} loss(y^{(i)}) + \lambda \text{regularizer}(w) \]
3. **develop a learning algorithm**
   \[ \arg\min_w \sum_{i=1}^{n} loss(y^{(i)}) + \lambda \text{regularizer}(w) \]
   Find \( w \) and \( b \) that minimize

---

**p-norms visualized**

- All \( p \)-norms penalize larger weights
  - \( p < 2 \) tends to create sparse (i.e., lots of 0 weights)
  - \( p > 2 \) tends to like similar weights

---

**p-norms visualized**

- Lines indicate penalty = 1
- For example, if \( w_1 = 0.5 \):
  - \( p = 1.0 \) penalty = 0.5
  - \( p = 1.5 \) penalty = 0.75
  - \( p = 2.0 \) penalty = 0.87
  - \( p = 3.0 \) penalty = 0.95
  - \( p = \infty \) penalty = 1
Minimizing with a regularizer

We know how to solve convex minimization problems using gradient descent:

\[ \text{argmin}_{\theta} \sum_{i=1}^{n} \text{loss}(y_i) \]

If we can ensure that the loss + regularizer is convex then we could still use gradient descent:

\[ \text{argmin}_{\theta} \sum_{i=1}^{n} \text{loss}(y_i) + \lambda \text{regularizer}(w) \]

make convex

Convexity revisited

One definition: The line segment between any two points on the function is above the function

Mathematically, \( f \) is convex if for all \( x_1, x_2 \)

\[ f(t x_1 + (1-t) x_2) \leq tf(x_1) + (1-t)f(x_2) \quad \forall 0 < t < 1 \]

the value of the function at some point between \( x_1 \) and \( x_2 \)

the value at some point on the line segment between \( x_1 \) and \( x_2 \)

Adding convex functions

Claim: If \( f \) and \( g \) are convex functions then so is the function \( z = f + g \)

Prove:

\[ z(tx_1 + (1-t)x_2) \leq tz(x_1) + (1-t)z(x_2) \quad \forall 0 < t < 1 \]

Mathematically, \( f \) is convex if for all \( x_1, x_2 \)

\[ f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2) \quad \forall 0 < t < 1 \]

Adding convex functions

By definition of the sum of two functions:

\[ z(tx_1 + (1-t)x_2) = f(tx_1 + (1-t)x_2) + g(tx_1 + (1-t)x_2) \]

\[ t f(x_1) + (1-t)f(x_2) + tf(x_1) + (1-t)f(x_2) + (1-t)g(x_1) \]

\[ = tf(x_1) + (1-t)f(x_2) + tg(x_1) + (1-t)g(x_2) \]

Then, given that:

\[ f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2) \]

\[ g(tx_1 + (1-t)x_2) \leq tg(x_1) + (1-t)g(x_2) \]

We know:

\[ f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2) \]

\[ g(tx_1 + (1-t)x_2) \leq tg(x_1) + (1-t)g(x_2) \]

So:

\[ z(tx_1 + (1-t)x_2) \leq tz(x_1) + (1-t)z(x_2) \]
Minimizing with a regularizer

We know how to solve convex minimization problems using gradient descent:
\[
\text{argmin}_{w,b} \sum_i \text{loss}(y_i')
\]
If we can ensure that the loss + regularizer is convex then we could still use gradient descent:
\[
\text{argmin}_{w,b} \sum_i \text{loss}(y_i') + \lambda \text{regularizer}(w)
\]
convex as long as both loss and regularizer are convex

p-norms are convex

\[
r(w,b) = \sqrt{\sum w_j^p} = \|w\|_p
\]
p-norms are convex for \( p \geq 1 \)

Model-based machine learning

1. pick a model
\[
0 = b + \sum_{j=1}^n w_j f_j
\]
2. pick a criteria to optimize (aka objective function)
\[
\sum_{i=1}^n \exp(-y_i (w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|_p^p
\]
3. develop a learning algorithm
\[
\text{argmin}_{w,b} \sum_{i=1}^n \exp(-y_i (w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|_p^p
\]
Find w and b that minimize

Our optimization criterion

\[
\text{argmin}_{w,b} \sum_{i=1}^n \exp(-y_i (w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|_p^p
\]
Loss function: penalizes examples where the prediction is different than the label
Regularizer: penalizes large weights
Key: this function is convex allowing us to use gradient descent
Gradient descent

- pick a starting point \( (w) \)
- repeat until loss doesn’t decrease in any dimension:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss [using the derivative]

\[
w_j = w_j - \eta \frac{d}{dw_j} (\text{loss}(w) + \text{regularizer}(w,b))
\]

\[
\arg\min_{w,b} \sum_{i=1}^{n} \exp(-y_i (w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|^2
\]

Some more maths

\[
\frac{d}{dw_j} \text{objective} = \frac{d}{dw_j} \sum_{i=1}^{n} \exp(-y_i (w \cdot x_i + b)) + \frac{\lambda}{2} \|w\|^2
\]

\[
= -\sum_{i=1}^{n} y_i x_i \exp(-y_i (w \cdot x_i + b)) + \lambda w_j
\]

Gradient descent

- pick a starting point \( (w) \)
- repeat until loss doesn’t decrease in any dimension:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss [using the derivative]

\[
w_j = w_j - \eta \frac{d}{dw_j} (\text{loss}(w) + \text{regularizer}(w,b))
\]

\[
w_j = w_j + \eta \sum_{i=1}^{n} y_i x_i \exp(-y_i (w \cdot x_i + b)) - \frac{\eta \lambda}{2} w_j
\]

The update

- learning rate direction to update
- constant: how far from wrong
- regularization

What effect does the regularizer have?
The update

If \( w_j \) is positive, reduces \( w_j \)
If \( w_j \) is negative, increases \( w_j \)

moves \( w_j \) towards 0

\[
 w_j = w_j + \eta y_i x_i \exp(-y_i (w \cdot x_i + b)) - \eta \lambda \text{sign}(w_j)
\]

L1 regularization

argmin \( \sum \exp(-y_i (w \cdot x_i + b)) + \lambda \|w\|_1 \)

\[
 \frac{d}{dw_j} \text{objective} = \frac{d}{dw_j} \sum \exp(-y_i (w \cdot x_i + b)) + \lambda \|w\|_1
\]

\[
 = \sum y_i x_i \exp(-y_i (w \cdot x_i + b)) + \lambda \text{sign}(w_j)
\]
Regularization with p-norms

L1:
\[ w_j = w_j + \eta (\text{loss}_{\text{correction}} - \lambda \text{sign}(w_j)) \]

L2:
\[ w_j = w_j + \eta (\text{loss}_{\text{correction}} - \lambda w_j) \]

Lp:
\[ w_j = w_j + \eta (\text{loss}_{\text{correction}} - \lambda c w_j^{p-1}) \]

How do higher order norms affect the weights?

Model-based machine learning

Develop a learning algorithm
\[
\text{argmin}_{w,b} \sum \exp(-y_i (w \cdot x_i + b)) + \frac{\lambda}{2} \| w \|^2
\]
Find w and b that minimize

Is gradient descent the only way to find w and b?

No! Many other ways to find the minimum.
Some are don’t even require iteration
Whole field called convex optimization

Regularizers summarized

L1 is popular because it tends to result in sparse solutions (i.e., lots of zero weights)
However, it is not differentiable, so it only works for gradient descent solvers

L2 is also popular because for some loss functions, it can be solved directly (no gradient descent required, though often iterative solvers still)

Lp is less popular since they don’t tend to shrink the weights enough

The other loss functions

Without regularization, the generic update is:
\[ W_j = W_j + \eta y_j x_j c \]

where
\[ c = \exp(-y_i (w \cdot x_i + b)) \quad \text{exponential} \]
\[ c = \mathbb{I}[y_i < 1] \quad \text{hinge loss} \]

\[ w_j = w_j + \eta (y_i - (w \cdot x_i + b)) x_j \quad \text{squared error} \]
Many tools support these different combinations:

Look at scikit learning package:


---

Common names:

- (Ordinary) Least squares: squared loss
- Ridge regression: squared loss with L2 regularization
- Lasso regression: squared loss with L1 regularization
- Elastic regression: squared loss with L1 AND L2 regularization
- Logistic regression: logistic loss